Master Thesis Proposal: Bluetooth Low Energy in the Contiki OS

Zhitao He, Shahid Raza (zhitao/shahid@sics.se)
Swedish ICT SICS

December 4, 2015

1 Motivation

Since its inclusion to the Bluetooth v4.0 standard in 2010, Bluetooth low energy (BLE) [1] has emerged as a strong contestant for weaving sensors and other small devices into the fast developing Internet of Things. Despite its later market entrance than IEEE 802.15.4 and proprietary low-power radio technologies, BLE has quickly gained traction, largely due to its natural, tight integration with classic Bluetooth in the same protocol stack. The latter's ubiquitous presence in PC's and smartphones nudge large hardware manufacturers to choose Bluetooth LE over rival technologies, when considering connectivity options for extending their products' reach to IoT.

The momentum for BLE-connected IoT is gathering, with the recent introduction of the Low-power IP feature, via the Internet Protocol Support Profile (IPSP) [2], to Bluetooth v4.2. This feature paves the way for direct IPv6/6LoWPAN connections between the devices and the Internet.

Today's BLE radio transceivers are integrated with a low-power MCU, often an ARM Cortex-M, in a system-on-chip, with few exceptions, apparently for cost and power-efficiency reasons. Chip vendors often provide a proprietary, free-of-charge BLE protocol stack, in the form of a precompiled and linked firmware library. Examples include Nordic Semiconductor's nRF51 and nRF52 series BLE SoCs and Texas Instruments' CC254x and CC265x series. The API is then usually around the Generic Attribute Profile (GATT) layer, above which application-specific profiles, e.g., a weight scale profile with weight scale and body composition services, are implemented by the application developer.

The open-source Contiki OS [3] for small devices has been the choice of many IoT-oriented research projects as well as a few commercial products, because of its compact, high-quality IPv6/6LoWPAN stack, which can be readily used together with the IoT routing protocol RPL [4] and application-layer protocols CoAP [5] and MQTT. Below the network layer, the dominant radio technology used by Contiki platforms is the PHY layer of the open standard IEEE 802.15.4, combined with a rich set of MAC protocols.

Birthplace of the Contiki project and still an active contributor, SICS Swedish ICT has a strong interest in adding BLE support to Contiki. In the Networked Embedded Systems Group, we have supervised two previous Master theses on BLE, one being a first Contiki port to the NordicSemi nRF51832 SoC [6] to study BLE's link-layer performance in comparison with IEEE 802.15.4 [7] [8].

2 Thesis Description

In this Master thesis, the student will port Contiki to the latest nRF52832 SoC, which is based on a Cortex-M4F MCU, a model with higher performance and better debugging support than its Cortex-M0-based predecessor. We aim at delivering a solid BLE research and development platform for our own research interests and potentially for the wider Contiki community. Another important goal of the thesis is exploring the interaction between application processes and the BLE firmware library, in order to identify performance bottlenecks and missing OS functionality. We hope the findings will bring enlightening ideas for a future open BLE stack.

We set a few main objectives for this thesis:

1. Implementation of the Contiki core
2. Implementation of a standard BLE application profile
3. Performance and power efficiency optimizations
4. (optional) IPv6/6LoWPAN support

2.1 Implementation of the Contiki core

To familiarize ourselves with the SDK and the GNU toolchain, we build a standalone sensing application with the BLE stack disabled. We program on top of the SDK's hardware abstraction layer. Because the nRF52 hardware development kit is form-compatible with Arduino Uno Rev.3 shields, we can use an analog sensor shield to provide the input signal. The application samples the signal and perform frequency analysis to detect abnormal events.

We then port a set of Contiki core system services and drivers to the platform.

- A clock source driver, that generates periodic timer interrupts from the Cortex-M4 HW timers.
- Digital output driver for LEDs
- Digital input driver for push buttons
- Contiki event timers, optionally reusing the tickless etimer from the previous thesis
- Serial output driver for printf()
- Serial input driver for the Contiki serial-line system service
- ADC driver for analog sensors
- Contiki realtime timers

We reimplement the sensing application on our Contiki system to verify the port is functional. A code size and performance comparison is made between the two implementations. A further bonus study would be an CPU instruction set performance comparison: one implementation would use the ARMv7 instruction set (as Cortex-M3), and the other would lend to the Cortex-M4F's DSP instructions and floating point unit.

2.2 Implementation of a BLE application profile

We use the GATT API of the SDK to implement two BLE application profiles.

- Implementation of a standard BLE application profile
- (optional) Design and implementation of a proprietary application profile for the sensing application built by ourselves.

We evaluate the implementation using the Nordic nRF Toolbox app for smartphones or a BLE USB dongle for PC's. We might want to use the ubertooth Bluetooth sniffer [9] to follow the connection events between the client and the server.

2.3 Performance and power efficiency optimizations

We use the nRF52 SDK's System on Chip library API to improve the performance and power efficiency of our Contiki port, by means of finer-grained provisioning of system resources (interrupts, power mode, etc) and streamlined coordination between the application processes and the BLE firmware library. We evaluate the results by measurement of task completion latencies and radio duty-cycle.

2.4 (optional) IPv6/6LoWPAN support

If we achieve the previous objectives early, we will try to add IPv6/6LoWPAN support to our Contiki port and run an example application on the IP stack.

3 Timeplan

A 20-week preliminary timeplan, subject to subsequent changes, is sketched as follows:

W1-2	Literature study
W3-4	Toolchain setup
W5-8	Contiki core
W9-12	BLE application
W13-16	Optimizations
W17-20	Thesis report

References

[1] Bluetooth SIG. Bluetooth Specification Version 4.2 [Vol 0]. Bluetooth Specification, December 2014. [`https://www.bluetooth.org/en-us/specification/adopted-specifications` Online; accessed 12-7-2015].

[2] T. Savolainen, K. Kerai, F. Berntsen, J. Decuir, R. Heydon, V. Zhodzishsky, and E. Callaway. Internet Protocol Support Profile. Bluetooth Specification, December 2014.

[3] Contiki OS website, 2015. [`http://www.contiki-os.org` Online; accessed 30-11-2015].

[4] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550, March 2012.

[5] Z. Shelby, K. Hartke, and C. Bormann. The Constrained Application Protocol (CoAP). RFC 7252 (Proposed Standard), June 2014.

[6] Nordic Semiconductor. nRF52832 datasheet, 2015.

[7] P.R. Narendra. Comparison of link layer of BLE and 802.15.4 - Running on Contiki OS. Master's thesis, KTH Royal Institute of Technology, Stockholm, Sweden, September 2014.

[8] A Contiki port for nRF51832, 2014. [`https://github.com/EarthLord/contiki` Online; accessed 3-12-2015].

[9] Project Ubertooth, 2015. [`http://ubertooth.sourceforge.net` Online; accessed 3-7-2015].